We normally publish the post-mortem for an assignment after it has been marked and released. Here is a list of common errors provided by the graders for assignment 4.

**Design Recipe**

- Types in contracts should always be capitalized. This includes user-defined types, such as `Unit-Type`, `Unit`, and `Train`.

- Where applicable, contracts should include user-defined types to be as specific as possible. For example, a function that consumes a `Unit-Type` should be represented with “Unit-Type” in the contract, and not simply “Sym”.

- Many students were missing examples for base case(s) throughout the assignment. When writing a recursive function, there should be an example for every base case, and an example for at least one of the recursive cases.

- Some students had examples and tests that did not abide by the contract for their functions, or by the data definitions of user-defined types. For example, if an example or test included a `Train` where each `Unit` did not have a unique serial number, that would not count as a valid test.

**Question 2**

- Some students did not use `contains?` as a helper function in part (c), which often lead to an incorrect implementation of the `has-duplicate?` function that only checks if consecutive elements in the consumed list are equal.

- Instead of using the `contains?` function directly in part (c), some students rewrote the function with a different name, which unnecessarily duplicates code.

**Question 3**

- In both parts (a) and (b), some students represented addition and subtraction by 3 using `add3` and `sub3` respectively. However, unless `add3` and `sub3` were defined prior to their use, it would be incorrect to use them, as `add3` and `sub3` are not built-in functions in Racket.

- Many students were missing a contract, or had an incorrect contract, for their template function.

- Some students combined the base case conditions into a single condition for the template function, instead of listing these conditions separately.

- Some students reversed the order of subtraction (i.e., `(- 3 n)` instead of `(- n 3)`) in the recursive case of the template function, which would not represent getting one step closer to the base cases for a `Nat3`.
Question 4

- In the data definition of a Unit, many students wrote that the first field of the Unit structure is a symbol, which was not specific enough. Instead, the type that would best represent the first field is Unit-Type.

- In the data definition for a Train, almost all students did not add the requirement that the serial numbers of each Unit must be unique.

- In part (c), some students did not check whether the consumed Train was empty, resulting in an error when taking the first of the consumed Train.

- In part (d), some students missed the case that ends-with-caboose? would produce false if there was more than one caboose in the consumed Train.

- In part (f), many students had blocks of repeated code that could have been written in a helper function instead. In particular, writing a helper function to check if a Unit was a car, and using headed-by? as a helper function would have reduced many blocks of repetitive code.

Ongoing Errors

The following is a list of common errors from previous assignments that were still repeated for assignment 4.

- Many students are still missing parameter references in their purpose statements. Purpose statements should meaningfully use each parameter name, and the parameter names should be written exactly as they appear in the function header.

- Unless otherwise specified, equal? should only be used to compare two values of unknown type. When the types of the arguments are known, use the most appropriate comparison function (e.g., symbol=?).

- Starting a new cond expression in an else clause is unneeded. Instead, directly check for the next condition in the original cond expression.